Network Working Group
Request for Comments: 80
NIC 5608

1 December 1970
E. Harslem
J. Heafner

PROTOCOLS AND DATA FORMATS

Because of recent discussions of protocols and data formats we issue this note to highlight our current attitudes and investigations in those regards. We first discuss some specific sequences, and then offer some thoughts on two general implementation approaches that will handle these and other specifics. We wish to place emphasis on the general solutions and not on the specifics.

INITIAL CONNECTION PROTOCOLS

We wish to make two points concerning specific Initial Connection Protocols (ICPs). Firstly, the ICP described in NEW/RFC #66--its generality and a restatement of that ICP. Secondly, a proposal for a variant ICP using basically the same logic as NWG/RFC #66.

I. NWG/RFC #66

The only technical error in this ICP is that as diagrammed both the Server and User send ALL messages before the connections are established which is inconsistent with Network Document No. 1. This can easily be remedied as will be shown in the restatement below.

In terms of generality, any ICP that is adopted as a standard should apply to more situations than a process calling a logger. That is, some Network service processes that hook directly to a user process, independent of logger action, could perhaps use a standard ICP. Thus, as is shown below, the process name field of the server socket should be a parameter with a value of zero being a special case for loggers.

Restatement of NWG/RFC #66 (using the same wording where appropriate)

1. To initiate contact, the using process attaches a receive socket (US) and requests connection to process SERV socket #1 in the serving HOST. (SERV = 0 for ICP to the logger.) As a result the using NCP sends:

1	4	3	1	1
RTS	US	SERV	1	P

over link 1, where P is the receive link.

2. The serving process (SERV) may decide to refuse the call, in which case it closes the connection. If it accepts the call, the serving process completes the connection (via an INIT system call, hence an STR).

1	3	1	4
STR	SERV	1	US

3. When the connection is completed, the user process allocates a nominal amount of space to the connection, resulting in the NCP sending:

1	1	4
ALL	P	SPACE

where SPACE is the amount.

4. The serving process then selects the socket pair it wishes to assign this user. It sends exactly an even 32 bit number over the connection. This even 32 bit number (SS) is the receive socket in the serving HOST. This socket and the next higher numbered socket are reserved for the using process.

5. It then closes the connection. The serving NCP sends (step 4):

4
SS

on link P, and (step 5):

1	3	1	4
CLS	SERV	1	US

on the control link (which is echoed by the using NCP).

6. Now that both server and user are aware of the remote socket pair for the duplex connection, <STR, RTS>s can be exchanged.

Server sends User

1	4	4	
STR	SS + 1	US	
RTS	SS	SS + 1	Q

where Q is the Server's receive link.

User sends Server

1	4	4	
STR	US + 1	SS	
RTS	US	SS + 1	R

where R is the User's receive link.

ALLocates may then be sent and transmission begun.

0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
A		
B		
C		
D		
E		
F		

II. A Variation of NWG/RFC #66

This variation reduces Network messages and eliminates duplication of information transfer.

Steps 3 and 4 above are deleted. The user process is not notified directly which of the Server's sockets it will be assigned. The user process, however, will listen on sockets US and US + 1 for calls from SERV after step 5 above. It can reject any spurious calls. In accepting the calls from SERV, the connection is established.

The following sample sequence illustrates this ICP. (The notation is as above.)

1. User → Server

1	4	3	1	1
RTS	US	SERV	1	P

2. Server → User

If accepted:

1	3	1	4
STR	SERV	1	US
CLS	SERV	1	US

If rejected:

1	3	1	4
CLS	SERV	1	US

3. If accepted, user listens on US and US + 1.

4. Server → User

1	4	4	
STR	SS + 1	US	
RTS	SS	US + 1	Q

5. User accepts the calls, hence:

User → Sender

1	4	4	
STR	US	SS + 1	
RTS	US + 1	SS	R

and the connection is established.

This reduces the number of network messages by two and only passes the information regarding the Server's sockets once via RTS and STR.

PRE-SPECIFIED DATA FORMATS

We would like to adopt those suggestions for data formats in NWG/RFC #42 and #63. We subscribe to multiple standards as solutions to particular problem classes.

AN ADAPTABLE MECHANISM

We would like to adapt to Network use, problem programs that were not planned with the Network in mind, and which, no doubt, will not easily succumb to Network standards existing at the time of their inclusion. This incompatibility problem is just as fundamental a part of the research underlying the Network as is different Host hardware. To require extensive front-ends on each

such program is not a reasonable goal. We view the Network as an amalgamation of a) Hosts that provide services; b) parasite Hosts that interface terminals to the services, and c) a spectrum of Hosts that behave as both users and providers of services. To require that each parasite Host handle different protocols and data formats for all services that its users need is not a reasonable goal. The result is programs and terminals that wish to communicate but do not speak the same language.

One approach to the protocol and data format problems is to provide an adaptable mechanism that programs and terminals can use to easily access Network resources. ARPA is sponsoring the Adaptive Communicator Project at Rand which is a research effort to investigate a teachable front-end process to interface man to program. The variety of terminal devices being explored include voice, tablets, sophisticated graphics terminals, etc.

The Adaptive Communicator looks very encouraging but it will not be ready for some time. The Network Project at Rand chose to take the adaptable approach (not adaptive, i.e., no heuristics, no self-learning). Our problem is to get Rand researchers onto the Network easily, assuming that they have different simultaneous applications calling for different program protocols and data configurations.

Protocols and data formats will be described separately to illuminate what we mean by adaptation. Protocols are sequences of "system calls" that correspond to (and result in NCP's issuance of) NCP commands. Data formats are the descriptions of regular message contents and are not meaningful to an NCP.

The Form Machine (adapting to data formats)

To put the reader in context, the Form Machine is of the class of finite state machines that recognize a form of regular expressions which, in our case, describe data formats. The notation, however, is aimed at particular descriptions and therefore can be more succinct, for our purposes, than the language of regular expressions.

The Form Machine is an experimental software package that couples a variety of programs and terminals whose data format requirements are different. We envision Form Machines located (to reduce Network traffic) at various service providing Hosts.

To test the Form Machine idea, we are implementing two IBM OS-callable subroutines; a compiler that compiles statements which describe forms or data formats; and an executor that executes a compiled form on a data stream.

To describe the Form Machine test, it is necessary to mention another program at Rand--the Network Services Program (NSP), which is a multi-access program that interfaces the Network Control Program both to arbitrary programs and to Video Graphics Consoles. (We view a terminal as just another program with a different interface, i.e., # characters/line, # lines/page, unique hardware features, the application to which it is put, etc.) The Form Machine subroutines are callable from NSP upon console or program direction.

Operationally, a console user names and specifies the data forms that he will use. The forms are compiled and stored for later use. At some future time when the user wishes to establish Network connections and transmit data, he dynamically associates named forms with each side of a port--a symbolically named Network full duplex connection. Data streams incoming or outgoing are executed according to the compiled form and the transformed data stream is then passed along to the console/program or to the Network, respectively.

The details of the syntax of our Form Machine notation are unimportant to the collective Network community. However, the provisions of the notation are of interest. It will eventually encompass the description of high performance CRT displays, TTY, and arbitrary file structures. To test its viability, a subset of such features is being implemented.

The current version is characterized by the following features:

1) Character code translation (viz., decimal, octal, hexidecimal, 8 bit ASCII, 7 bit ASCII, EBCDIC, and binary).

2) Multiple break strings (many terminals have multiple termination signals).

3) Insertion of literals (used primarily for display information presentation).

4) Skip or delete arbitrary strings (used to remove record sequence numbers, etc., that are not to be displayed).

5) Record sequence number generation.

6) String-length computation and insertion.

7) Arbitrary data string length specifications, e.g., "a hex literal string followed by an arbitrary number of EBCDIC characters, followed by a break string, ...".

8) Concatenation of Network messages, i.e., the execution of compiled forms on incomplete data strings.

9) Data field transposition.

10) Both explicit and indefinite multiplicative factors for both single and multi-line messages.

Features that are not being implemented but will be added, if successful, include:

1) Graphics oriented descriptions.

2) General number translations.

3) Conditional statements.

4) A pointer capability.

The Protocol Manager (adapting to NCP command sequences)

The NSP allows terminal users and programs to work at the NCP protocol level; i.e., LISTEN, INIT, et al. It also allows them to transmit and massage information meaningful only to themselves. This "hands-on" approach is desirable from the systems programmer's, or exploratory point of view. However, it is desirable to eliminate the laborious "handshaking" for the researcher who repeatedly uses a given remote program by allowing him to define, store, retrieve, and execute "canned" protocol sequences.

We are currently specifying a Protocol Manager as a module of NSP that will allow the above operations on NCP command sequences. Features of the module are:

1) The sequences may contain "break points" to permit the console user to dynamically inject any contextually needed information.

2) The parameters of a command may contain tokens whose values are supplied by the remote party during the protocol dialog. For example, in Note #66 the socket number provided by the server is to be used by the user in subsequent RTS, STR commands.

REQUEST

We would like to hear from anyone concerning the notion of adaptation to data formats and protocol. Is this a reasonable approach? What should it encompass?

JFH:EFH:hs

Distribution

Albert Vezza, MIT
Alfred Cocanower, MERIT
Gerry Cole, SDC
Bill English, SRI
Bob Flegel, Utah
James Forgie, LL
Peggy Karp, MITRE
Nico Haberman, Carnegie-Mellon
John Heafner, RAND
Bob Kahn, BB&N
Margie Lannon, Harvard
James Madden, Univ. of Ill.
Thomas O'Sullivan, Raytheon
Larry Roberts, ARPA
Robert Sproull, Stanford
Ron Stoughton, UCSB
Chuck Rose, Case University
Benita Kirstel, UCLA